Improved Bounds for Learning Symmetric Juntas

*Richard J. Lipton** *Evangelos Markakis** *Aranyak Mehta**

1 Introduction

We consider a fundamental problem in computational learning theory: learning in the presence of irrelevant information. In particular we are interested in learning an arbitrary boolean function of n variables which depends only on k of them. The problem has a lot of interesting applications in artificial intelligence, neural networks and machine learning theory.

The problem was first proposed by Blum [1] and Blum and Langley [2]. Ever since, the first nontrivial algorithm was given by [4], which runs in time $n^{0.7k}poly(\log 1/\delta, 2^k, n)$, for general juntas and $n^{\frac{2}{3}k}poly(\log 1/\delta, 2^k, n)$ for symmetric juntas. We give an algorithm for symmetric juntas which runs in time $n^{k/3(1+o(1))}poly(\log 1/\delta, 2^k, n)$. We further show that when k is bigger than some large enough constant, the algorithm runs in time $n^{0.18k}poly(\log 1/\delta, 2^k, n)$. To our knowledge, this is the best known upper bound for learning symmetric juntas under the uniform distribution. The same algorithm has also been proposed by [5]. In [5] it was shown that the running time is bounded by $n^{k/2(1+o(1))}poly(\log 1/\delta, 2^k, n)$. It was also shown that under a certain number theoretic assumption, the running time is $n^{k/3}poly(\log 1/\delta, 2^k, n)$.

2 Notation

We consider the PAC learning model introduced by Valiant [6]. In this model a concept class $\mathcal{C} = \bigcup \mathcal{C}_n$ is a collection of functions indexed by some parameter n (in our case $\mathcal{C}_n$ is a subset of boolean functions on n bits, which we will define later on). Let $f \in \mathcal{C}_n$ be an unknown target function. Let $\mathcal{D}$ be a probability distribution over the domain of f. A learning algorithm $\mathcal{A}$ for $\mathcal{C}$ has access to an oracle, which when queried outputs a random labeled example $< x, f(x) >$, where x is drawn at random according to the probability distribution $\mathcal{D}$ and $f(x)$ is the value of f at x. The algorithm $\mathcal{A}$ outputs a hypothesis h, which is a function from $\mathcal{C}$. We define the error of the hypothesis to be $error(h) = Pr[h(x) \neq f(x)]$, where x is drawn according to $\mathcal{D}$.

*Address: College of Computing, Georgia Institute of Technology, Atlanta, GA 30332–0280. E-mail: {rjl,vangelis,aranyak}@cc.gatech.edu. Research supported by NSF grant CCR-0002299.

Definition 1 *An algorithm $\mathcal{A}$ is a PAC learning algorithm for $\mathcal{C}$ under the distribution $\mathcal{D}$ if for any $f \in \mathcal{C}$ and for any $\epsilon, \delta > 0$, whenever $\mathcal{A}$ is given access to an oracle that outputs labeled examples of f according to the distribution $\mathcal{D}$, then with probability $1-\delta$, $\mathcal{A}$ outputs a hypothesis $h \in \mathcal{C}$ such that $error(h) \leq \epsilon$. The probability is taken over the random examples of the oracle and over any internal randomization of the algorithm.*

A boolean function $f : \{0,1\}^n \to \{0,1\}$ on n variables is a symmetric function if for any permutation $\pi \in S_n$, $f(x_1, ..., x_n) = f(\pi(x_1), ..., \pi(x_n))$. Hence the value of f at $(x_1, ..., x_n)$ depends only on the *weight* of $(x_1, ..., x_n)$, which is the number of variables that are set to 1. We will often describe a symmetric boolean function on n variables by a $(n+1)$-bit string $f_0 f_1 ... f_n$, where f_i is the value of f when i variables are set to 1.

Given a boolean function f on n variables $x_1, ..., x_n$, we will say that x_i is a *relevant* variable for f if there exist $x, y \in \{0,1\}^n$ which differ only in the i-th coordinate and $f(x) \neq f(y)$. We will call f a k-junta if f has at most k relevant variables. The concept class we consider in this paper is the class of symmetric k-juntas. In particular $\mathcal{C}_n = \bigcup \mathcal{C}_{n,k}$ and $\mathcal{C}_{n,k} = \{f : \{0,1\}^n \to \{0,1\}$ s.t. f is a symmetric k-junta$\}$. Therefore each $f \in \mathcal{C}_{n,k}$ can be described by a $(k+1)$-bit string $f_0 f_1 ... f_k$ specifying the value of f at each possible weight of its relevant variables.

In our case, the probability distribution of the oracle will be the uniform distribution over $\{0,1\}^n$ and we will set the accuracy parameter ϵ to be 0. Hence we want to find the actual target function f.

3 The algorithm

Below we describe the algorithm. Throughout the algorithm, we maintain a set of relevant variables, R.

- Set $R := \emptyset$.
- Check if the function is constant.
- If not, then check if f is a parity function using the algorithm of [3].
- If f is not parity or constant then for every t starting from $t = 1$ do:
  - For every subset of t variables, say $S = \{x_{i_1}, ..., x_{i_t}\}$ compute the following $t+1$ probabilities:

$$Pr[(x_{i_1}, ..., x_{i_t}) = u_j \mid f(x) = 1], \quad u_j = 1^j 0^{t-j}, j = 0, ..., t$$

  - If any of the above probabilities are not equal to $\frac{1}{2^t}$, then include the set S in the relevant variables, $R := R \cup S$. Do this for every S of size t and then halt.
  - If for all sets S of size t the probabilities are all equal to $\frac{1}{2^t}$ then $t := t + 1$ and repeat the procedure.

Before proceeding to the analysis of the algorithm we would like to make some observations:

Remark 1 The choice of the strings $u_j, j = 0, ..., t$ is not unique. Since the target function is symmetric, any collection of $t + 1$ strings with distinct weights from 0 to t would be sufficient, as will become clear from our analysis.

Remark 2 The algorithm may as well compute the probabilities:

$$Pr[f(x) = 1 \mid x_{i_1} = b_1, ..., x_{i_t} = b_t] \text{ for } b_1, ..., b_t \in \{0, 1\}.$$

This is because one can easily see that these probabilities are the same (over different choices of the bits b_i) if and only if the probabilities that the algorithm considers are the same.

Remark 3 We also have the following: the probabilities considered by the algorithm are the same (and equal to $\frac{1}{2^t}$) for each $l \leq t$ if and only if all Fourier coefficients of size $l \leq t$ are 0. This can be proved by using the fact that the Fourier coefficients are simply linear combinations of the probabilities defined in the first remark, with coefficients $+1$ and -1. The forward implication is immediate, while the backward implication can be proved by induction. We will include the proof in the final version of the paper. Hence the algorithm is equivalent to the Fourier-based learning algorithm [4][Section 3.2].

3.1 Analysis

We can decide whether the target function f is constant or not with confidence $1 - \delta$ in time $poly(\log 1/\delta, 2^k)$ [4]. Furthermore, if f is a parity function we can learn it in time $n^\omega poly(\log 1/\delta, 2^k)$ by the algorithm of [3] and the argument in Proposition 10 of [4]. Here ω is the exponent for matrix multiplication, $\omega < 2.376$.

It is not difficult to see that we can estimate the conditional probabilities involved in the algorithm and decide with confidence $1 - \delta$ if they are equal to $\frac{1}{2^t}$ using standard Chernoff-Hoeffding bounds as in [4]. We omit the details for the final version:

Lemma 4 *For any $t \leq k$ and for any set of variables S, we can decide with confidence $1 - \delta$ if any of the conditional probabilities $Pr[(x_{i_1}, ..., x_{i_t}) = u_j \mid f(x) = 1]$ are not equal to $\frac{1}{2^t}$ in time $poly(\log 1/\delta,\ 2^k, n)$.*

We also need to show that the very first time that some of the above conditional probabilities are not equal to $\frac{1}{2^t}$, the corresponding set S is indeed a set of relevant variables. Clearly if S contained only irrelevant variables then the probabilities will be exactly $\frac{1}{2^t}$. Hence it suffices to show:

Lemma 5 *Let $t > 0$ and suppose that there exists a set of variables $S = \{x_{i_1}, ..., x_{i_t}\}$ and an index $j \in \{0, ..., t\}$ such that*

$$Pr[(x_{i_1}, ..., x_{i_t}) = u_j \mid f(x) = 1] \neq \frac{1}{2^t}$$

Let $S = S_1 \cup S_2$, $|S_1| = t_1, |S_2| = t_2$, where S_1 is the set of relevant variables of S and S_2 is the set of irrelevant variables of S. Then if u'_j is the projection of u_j to S_1 and $S_1 = \{x_{i_1}, ..., x_{i_{t_1}}\}$, we have:

$$Pr[(x_{i_1}, ..., x_{i_{t_1}}) = u'_j \mid f(x) = 1] \neq \frac{1}{2^{t_1}}$$

Proof : Since the variables of S_2 are irrelevant we have:

$$Pr[(x_{i_1}, ..., x_{i_t}) = u_j \mid f(x) = 1] = \frac{1}{2^{t_2}} Pr[(x_{i_1}, ..., x_{i_{t_1}}) = u'_j \mid f(x) = 1] \neq \frac{1}{2^t}$$

Therefore:

$$Pr[(x_{i_1}, ..., x_{i_{t_1}}) = u'_j \mid f(x) = 1] \neq \frac{1}{2^{t_1}}$$

□

Since the function is symmetric, the first time we will identify some relevant variables, we will actually be able to identify all the relevant variables. This is so because all t-size subsets of relevant variables will give exactly the same probabilities. Since at each step we look at $\binom{n}{t}$ subsets of variables, if we can prove that there exists a t for which at least one of the above probabilities is not equal to $\frac{1}{2^t}$ for some subset of relevant variables, then the running time of the algorithm will be $n^t poly(\log 1/\delta, 2^k, n)$.

3.1.1 A bound of $k/3$

In this section we prove that for any target function, the algorithm terminates for $t \leq \frac{k}{3}(1 + o(1))$. Suppose on the contrary that there exists a function $f = f_0 f_1 ... f_k$ such that for any $t \leq \frac{k}{3}(1 + o(1))$, the probabilities computed during the algorithm are equal. It is enough to show that f has to be either a constant or a parity function. It is easy to see that for any t, for any set S of t relevant variables $S = \{x_{i_1}, ..., x_{i_t}\}$, and for any $j \in \{0, 1, ..., t\}$ the following holds:

$$Pr[(x_{i_1}, ..., x_{i_t}) = u_j \mid f(x) = 1] = \frac{\sum_{i=0}^{k} f_i \binom{k-t}{i-j}}{\sum_{i=0}^{k} f_i \binom{k}{i}}$$

We have made the standard assumption that a binomial coefficient $\binom{\alpha}{\beta}$ is 0 whenever $\beta > \alpha$ or $\beta < 0$. The fact that for the function f all the above probabilities are equal to $\frac{1}{2^t}$ means that for $t \leq \frac{k}{3}(1 + o(1))$:

$$\sum f_i \binom{k-t}{i} = \sum f_i \binom{k-t}{i-1} = ... = \sum f_i \binom{k-t}{i-t} = \frac{\sum f_i \binom{k}{i}}{2^t} = c$$

Pick primes $p, q \in [k/3 - o(k), k/3]$, $p > q$. This can be done because of the prime number theorem. Let $t = k - 2p = \frac{k}{3}(1 + o(1))$. Consider the sums mod p.

	p	q
0	0	0
1	$x0^p\bar{x}1^l$	$a0^q\bar{a}1^m$
2	$x\bar{x}xw$	$a\bar{a}ad$
3	$x1^p\bar{x}0^l$	$a1^q\bar{a}0^m$
4	1^k	1^k

Table 1: Different cases for mod p and mod q

Since $k-t=2p$, the only binomial coefficients not 0 mod p are $\binom{k-t}{0} \equiv 1, \binom{k-t}{p} \equiv 2$ and $\binom{k-t}{k-t} \equiv 1$. Hence the sums mod p are $f_0+2f_p+f_{2p} \equiv f_1+2f_{p+1}+f_{2p+1} \equiv ... \equiv f_t+2f_{p+t}+f_{2p+t} \equiv c \equiv c_p \ mod\ p$, where $0 \leq c_p \leq p-1$. Since each f_i is either 0 or 1 and p is large enough, these sums are actually equal to c_p. Let us consider all the possibilities for c_p. For ease of notation we will write the string $f=f_0f_1...f_k$ as $xyzw$, where x,y,z are strings of length p each and w is of length $l=k-3p=o(k)$ (note that l can be 0).

1. $c_p=0$. In this case $f_i=0$ for all i, and we are done, because f has to be the constant function 0.

2. $c_p=1$. The first equation gives $f_0+2f_p+f_{2p}=1$. This can only be if $f_p=0$, and $f_0=\bar{f}_{2p}$. By looking at the remaining sums, we obtain that $f=x0^p\bar{x}1^l$, and the first l bits of x are all 1.

3. $c_p=2$. The first equation gives $f_0+2f_p+f_{2p}=2$. This can only be either if $f_0=f_{2p}=0$, and $f_p=1$, or if $f_0=f_{2p}=1$, and $f_p=0$. By looking at the rest of the equations we have that $f=x\bar{x}xw$, where w consists of the first l bits of $\bar{x}$.

4. $c_p=3$. In the same manner we get $f=x1^p\bar{x}0^l$, and the first l bits of x are 0.

5. $c_p=4$. Here we immediately have that $f=1^k$, and we are done.

By considering the sums mod q, we can obtain similar expressions for f. We summarize all the cases in Table 1. The second column is the expression for f for the various values of the sums mod p, and the third column is for the values of the sums mod q. Here x is a string of length p, $l=k-3p=o(k)$ and w is the length l prefix of $\bar{x}$; a is a string of length q, $m=k-3q=o(k)$ and d is the length m prefix of $\bar{a}$.

Since the case of 0 or 4 mod p or q leads to a constant function, we are left with 9 cases: c = 1, 2 or 3 mod p and c = 1, 2, or 3 mod q. We will denote by Case i,j the case where $c=i\ mod\ p$ and $j\ mod\ q$.

Case 1,1:

Observe first that

$$f_{i+2p}=\bar{f}_i, 0 \leq i \leq k-2p \tag{1}$$

$$f_{i-2q} = \bar{f}_i, 2q \le i \le k. \tag{2}$$

Let $g = p - q$. From Table 1, $f_i = 0$, for $p \le i \le 2p-1$. Then 2 implies that $f_i = 1$, for $0 \le i \le 2g-1$. Then 1 implies that $f_i = 0$, for $2p \le i \le 2p+2g-1$. Using 2 again, we get $f_i = 1$, for $2g \le i \le 4g-1$. Thus using 1 and 2 alternately, we eventually get $f_{3q} = 0$. But this cannot be because f_{3q} is 1 in this case.

Case 1,2:
Note that now the following equation holds:

$$f_{i+q} = \bar{f}_i,\ 0 \le i \le k-q. \tag{3}$$

Since $c_p = 1$, $f_p = 0$. By Equation 3 we know that $f_{p+q} = \bar{f}_p = 1$. But $p + q = 2p - g$. By the mod p equations, $f_{2p-g} = 0$, a contradiction.

Case 1,3:
From the mod p equations $f_p = 0$, but from the mod q equations $f_p = f_{q+g} = 1$, a contradiction.

Case 2,1 :
By the mod q equations we have that $f_i = 1$ for $3p - g \le i \le 3p - 1$. By using the mod p equations, this implies that $f_i = 1$ for $p - g \le i \le p - 1$. However again by the mod q equations we have that $f_i = 0$ in that range, hence a contradiction.

Case 2,2 :
In this case, it is easy to see that $|i - j| = p$ or $|i - j| = q$ implies that $f_i = \bar{f}_j$. We claim that f is a parity function. Let α, β be two integers such that $\alpha p + \beta q = 1$. Without loss of generality, assume that $\alpha > 0, \beta < 0$. Let $\gamma = -\beta > 0$. Consider the following game: We start at some index $i < k$, where the value of the function is f_i. We are allowed to move p steps to the right, or q steps to the left, as long as we stay inside $\{0, 1, ..., k\}$. On every move, the value of the function at the new position is the complement of the value at the start position. We are allowed to make at most α moves to the right, and at most γ moves to the left. We claim that we can make exactly α right moves and γ left moves, at the end of which we end up at index $i + 1$. This is achieved through the following algorithm: start at i; make a right move if $i \le k - p$ and we have not exhausted all the α right moves allowed; make a left move if $i \ge q$ and we have not exhausted all the γ left moves. Consider the first time the algorithm cannot make a move. This can happen because of three reasons: (1) it has exhausted all its right and left moves. In this case the current index must be $i + 1$. (2) the current index j is greater than $k - p$ and we have exhausted all the γ left moves and we have $\alpha' > 0$ right moves remaining. This cannot happen because then $i+1 = j + \alpha' p \ge j + p > k$. (3) the current index j is less than q and we have exhausted all the α right moves and we have $\gamma' > 0$ left moves remaining. This cannot happen because then $i + 1 = j - \gamma' q \le j - q < 0$. A fourth reason could be that before exhausting either the left or the right moves we cannot move left or right. But since p and q are less than $k/2$, this case cannot occur.

Hence we have made $\alpha + \gamma$ moves in total, and we have reached index $i + 1$. It is easy to see that $\alpha + \gamma$ is odd. Since on every move we complement the value of f, $f_{i+1} = \bar{f}_i$ and therefore f is a parity function.

Case 2,3 :
This is similar to the Case 2,1.

Case 3,1 :
From the mod p equations $f_p = 1$, but from the mod q equations $f_p = f_{q+g} = 0$, a contradiction.

Case 3,2 :
Note that Equation 3 is true in this case as well. From the mod p equations $f_p = 1$. By using the mod q equations we know that $f_{p+q} = \bar{f}_p = 0$. But $p + q = 2p - g$ and by the mod p equations, $f_{2p-g} = 1$, a contradiction.

Case 3,3 :
This is symmetric to Case 1,1.

Thus we have proved that the only cases possible are the the cases (0,0), (2,2) and (4,4), and thus the function has to be constant or parity.

Hence we have the following theorem:

Theorem 6 *We can learn symmetric k-juntas under the uniform distribution in time* $n^{\frac{k}{3}(1+o(1))}poly(\log 1/\delta, 2^k, n)$.

3.1.2 A better bound

We now prove that the algorithm will in fact terminate for $t \leq 0.18k$, for large enough k.

Theorem 7 *For large enough k, we can learn symmetric k-juntas under the uniform distribution in time* $n^{0.18k}poly(\log 1/\delta, 2^k, n)$.

Proof :

It is enough to show that if for $t \leq 0.18k$ all the probabilities computed by the algorithm are exactly $\frac{1}{2^t}$ then the target function is either a constant or a parity function.

Suppose there exists a function $f = f_0 f_1 ... f_k$ on k bits (which is not a constant or a parity function) for which the algorithm does not terminate for $t \leq 0.18k$. Then the following sums of binomial coefficients are equal, for $t = 1, ..., 0.18k$:

$$\sum f_i \binom{k-t}{i} = \sum f_i \binom{k-t}{i-1} = ... = \sum f_i \binom{k-t}{i-t} = \frac{\sum \binom{k}{i}}{2^t} = c \qquad (4)$$

We will show that this is possible only for the constant or the parity functions. Pick a prime $p \in (\frac{k}{19.5}, \frac{k}{19}]$. We know that such a prime exists for large enough k by the prime number theorem. By the choice of p, we have $k = 19p + r_p$ for some $r_p \geq 0$. Let $t = k - 16p$. Clearly $t \leq 0.18k$ and equation 4 holds with $k - t = 16p$. The sums in 4 are also congruent *mod p*.

In particular the following 3 sums are congruent $mod\ p$:

$$\sum f_i \binom{16p}{i} \equiv \sum f_i \binom{16p}{i-p} \equiv \sum f_i \binom{16p}{i-2p} \equiv c\ modp \tag{5}$$

It is well known that the binomial coefficients of the form $\binom{\alpha p}{i}$ are 0 $mod\ p$ when i is not a multiple of p and equal to $\binom{\alpha}{j} mod\ p$ if $i = jp$. Therefore the sums above, when taken $mod\ p$ will involve only terms in which i is a multiple of p. For example the first sum in 5 will be congruent to:

$$\sum f_i \binom{16p}{i} \equiv f_0 \binom{16}{0} + f_p \binom{16}{1} + f_{2p} \binom{16}{2} + ... + f_{16p} \binom{16}{16}\ mod\ p$$

Similarly the second sum in 5 will be:

$$\sum f_i \binom{16p}{i-p} \equiv f_p \binom{16}{0} + f_{2p} \binom{16}{1} + f_{3p} \binom{16}{2} + ... + f_{17p} \binom{16}{16}\ mod\ p$$

Consider the following 18-bit symmetric function g defined as $g_0 = f_0, g_1 = f_p, g_2 = f_{2p}, ..., g_{18} = f_{18p}$. By equation 5 and the above observation we have that:

$$\sum g_i \binom{16}{i} \equiv \sum g_i \binom{16}{i-1} \equiv \sum g_i \binom{16}{i-2} \equiv c\ modp$$

If we pick k to be large enough, e.g. $k \geq 20 \cdot 2^{16}$, then $p > 2^{16}$ and the sums in the last congruence are actually equal. By doing a computer search on all 18-bit symmetric functions we found that the only functions with this property are the constant and the parity functions.

This means that the subsequence of f consisting of the levels $0, p, 2p, ..., 18p$ is either constant or parity. We can repeat this argument for the subsequence $1, p+1, 2p+1, ..., 18p+1$, then the subsequence $2, p+2, 2p+2, ..., 18p+2$, and continue up to the levels $p+r_p, 2p+r_p, 3p+r_p..., 18p+p+r_p = k$. Thus we obtain that each subsequence is either constant or forms a parity function.

In fact, either all the subsequences above are the constant sequence 1, or they are all the constant sequence 0, or they are all a parity function (not necessarily the same parity, they could be a mix of odd and even parities). This is because all the sums are equal to the same value c.

We now pick another prime $q \in (\frac{k}{19.5}, \frac{k}{19}]$. Clearly $k = 19q + r_q$, for some $r_q \geq 0$. Let $t = k - 16q$. Since $t \leq 0.18k$, equation 4 holds again with $k - t = 16q$. The sums in equation 4 will involve binomial coefficients of the form $\binom{16q}{i}$ and we will consider congruences $mod\ q$. In particular we have:

$$\sum f_i \binom{16q}{i} \equiv \sum f_i \binom{16q}{i-q} \equiv \sum f_i \binom{16q}{i-2q} \equiv c\ mod\ q \tag{6}$$

We can define the following 18-bit symmetric function h, with $h_0 = f_0, h_1 = f_q, h_2 = f_{2q}, ..., h_{18} = f_{18q}$. Again, if k is large enough we will have:

$$\sum h_i \binom{16}{i} = \sum h_i \binom{16}{i-1} = \sum h_i \binom{16}{i-2}$$

This means that the subsequence of f consisting of the levels $0, q, 2q, ..., 18q$ is either constant or parity. We can repeat this argument for the subsequence $1, q+1, 2q+1, ..., 18q+1$, and continue up to the levels $q+r, 2q+r, 3q+r..., 18q+q+r = k$. Just as for the *mod p* case we get that all the subsequences above are the constant sequence 1, or they are all the constant sequence 0, or they are all a parity function. It is also obvious that when the subsequences in the mod p case are a constant sequence, the subsequences in the mod q case have to be the same constant sequence.

Hence there are only three cases to consider.
Case 1: All the subsequences are the constant sequence 0. Then f is the constant function 0.
Case 2: All the subsequences are the constant sequence 1. Then f is the constant function 1.
Case 3: Each subsequence is a parity. Then $|i-j| = p$ or $|i-j| = q$ implies that $f_i = \bar{f}_j$. Since p and q are less than $k/2$, this case is identical to the case 2,2 in Section 3.1.1. Thus f is a parity function.

This completes the proof.

□

4 Discussion

The proof of $0.18k$ is scalable, i.e. a better building block would imply a better bound. We still do not know what is the best building block though. However the bound we can obtain with this method can be only a constant fraction of k. It seems that the technique cannot give even $o(k)$, since the building block is explicit.

References

[1] A. Blum. Relevant examples and relevant features: Thoughts from computational learning theory. *AAAI Fall Symposium on Relevance*, 1994.

[2] A. Blum, P. Langley. Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97, pp. 245-271, 1997.

[3] D. Helmbold, R. Sloan, M. Warmuth. Learning integer lattices. *SIAM Journal on Computing*, 21(2), 240-266, 1992.

[4] E. Mossel, R. O'Donnel, R. Servedio. Learning Juntas. *STOC*, 2003.

[5] N. Vishnoi, N. Devanur, R. Lipton. Learning Symmetric Juntas. manuscript, 2003.

[6] L. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11), pp. 1134-1142, 1984.